UML Overview

Based in parts on ‘UML Distilled’ from Martin Fowler

Alexander Egyed

CSCI 612

March, 1999

Outline

• UML Views
• Diagrams
  – Use Case Diagrams
  – Class Diagrams
  – Interaction Diagrams
  – Package Diagrams
  – State Diagrams
  – Activity Diagrams
  – Deployment Diagrams
• Some Conclusions
Why Modeling?
- Programs become more complex
- Failures more costly
- Edit-and-Compile cycle not very efficient

Why New Models for OO?
- Paradigm shift from *Functional Oriented Design* to *Object-Oriented Design* => functional modeling leads to functional decomposition
- New Concepts such as *Inheritance*, *Polymorphism*, *Data/Behavior Encapsulation*, etc.

### UML History

- **OMT** (Rumbaugh and others) 1995
- **Booch**
- **UML 0.8** 1995
- **UML 0.9** 1996
- **UML 1.1** (OMG Standard) 1997
- **OOSE** (Jacobson and others)
- **UML 1.3** (beta) 1999
UML 1.1 OMG Standard

UML is a language for specifying, constructing, and documenting software systems:

- General Purpose Modeling Language
- Merges Modeling Element from Booch, Rumbaugh, Jacobson and others
- Object-Oriented Analysis and Design
- Graphical Notation supporting numerous diagrams
- Extensible Notation (Stereotypes)
- Extensible Semantics (Object Constraint Language)

UML is backed by numerous software producers such as Digital, HP, IBM, Oracle, Microsoft, Unisys, and others

However, ...
- Semantics often ambiguous (only the UML Notation Meta-Model well defined)
- As of yet there is no tool that supports all of UML (not even Rational Rose)
- Uses both OO and functional development concepts

Nevertheless, ...
- Common model eases communication and interaction
- More precise meaning (semantics) can be added (customized)
Views in UML

Components and Connectors

Components:
- Classes, Objects, and Packages
- States and Activities
- Actors and Use-Cases

Connectors:
- Basically three types: Communication (control and data), Containment, Attachment
- Restricted in what types of components they link
- First class citizens
Outline

• UML Views
• Diagrams
  – Use Case Diagrams
  – Class Diagrams
  – Interaction Diagrams
  – Package Diagrams
  – State Diagrams
  – Activity Diagrams
  – Deployment Diagrams
• Some Conclusions

Use-Case Diagrams

- Capture those parts of the system that are visible to the outside (e.g. users or other systems).
- A use case is usually about a concrete goal or task from the user’s point of view
- Use-Cases may give no feedback in terms of complexity or system decomposition

=> mainly an analysis method.
Outline

- UML Views
- Diagrams
  - Use Case Diagrams
  - Class Diagrams
  - Interaction Diagrams
  - Package Diagrams
  - State Diagrams
  - Activity Diagrams
  - Deployment Diagrams
- Some Conclusions
Class Diagrams

Class Diagrams

- Describe the static relationship of classes in a system. These relationships must always be true.
- Classes often represent physical or otherwise tangible entities but this must not always be true, either.
- Classes are probably the most misunderstood and misused elements in OO design.

Frequently used Connectors:
- Associations
- Aggregation
- Generalization
Class Diagrams

Classes are usually the most central OO development elements since they reflect the actual implementation the closest -> this may have negative side effects.

Need to distinguish three types of classes:
- Conceptual Classes: Should capture domain entities with no regard to the actual implementation.
- Design Classes: Should capture design entities without overly committing to implementation details
- Implementation Classes: Should capture the physical model (e.g. strongly influenced by programming language)

Domain Classes

Domain Models show real-world objects and the relationships between them.
- Constraint Rules: e.g. Precondition \{Amount = 0\}. Also useful to model post conditions and invariants (Design by Contract)

- Multiple Classifications

![Class Diagram]

- Dynamic Classifications

![Class Diagram]

- Aggregation and Composition: Aggregation is the part-of relationship. Easy? Consider: Employee part-of Company, Engine part-of Car, etc. Sounds right but may not be true!

![Class Diagram]
Advanced Concepts in Classes

- Derived Associations and Attributes
- Interfaces and Abstract Classes (generalization vs. realization)
- Reference Classes vs. Value Classes
  => not all classes are equal
  (many customers, one date)
- Classification vs. Generalization (is-a is dangerous)
  1) Peter is-a Doctor; 2) Doctor is-a Employee;
  3) Doctor is-a Profession => Peter is-a Profession?

Advanced Concepts in Classes

- Qualified Associations
- Association Classes

Implies only one instance of Employment per Association of Person and Company

=> Person employed by same company at different times?
Advanced Concepts in Classes

- Parameterized Class

- Instantiated Class, Utility Class, Meta-Class, ...
- Visibility (public, private, protected)

- Object Diagrams?

Outline

- UML Views
- Diagrams
  - Use Case Diagrams
  - Class Diagrams
  - Interaction Diagrams
  - Package Diagrams
  - State Diagrams
  - Activity Diagrams
  - Deployment Diagrams
- Some Conclusions
Interaction Diagrams

- Describe how groups of objects interact with each other. Interaction diagram show the behavior of objects during a particular time or time frame.
- On a conceptual level, interaction diagrams may show the collaboration of use cases and conceptual classes -> things the user and customer can relate to.
- On a specification and implementation level the focus is on the collaboration of the software system components.

Two basic types
- Sequence Diagrams
- Collaboration Diagrams

Sequence Diagram

[Diagram showing interactions between Building Owner and Sections, including operations like create lease, add zone to section, search, and conditions for lease existence with iteration and object life cycle.]
Concurrent Processes and Activations

USC - Center for Software Engineering

Asynchronous Message

Synchronous Message

Other Processing Suppressed

Object deletes itself

Object is killed

new a Transaction

new a Transaction Coordinator

new a first Transaction Checker

new a second Transaction Checker

all done?
ok

all done?

beValid

beInvalid

fails

Kills other checkers
Sequence Diagrams:
-> easy to see the sequence of event happening

Collaboration Diagrams:
-> easy to see static connection (configuration) of objects

- Diagrams emphasize simplicity. It is easy to create and to understand them as long as the process depicted is sequential in nature (one use case).
- Both types of interaction diagrams tend to become awkward when conditions and loops are used (more generalized behavior).
- Conditional behavior is best represented through separate diagrams.
Outline

USC - Center for Software Engineering

• UML Views
• Diagrams
  – Use Case Diagrams
  – Class Diagrams
  – Interaction Diagrams
  – Package Diagrams
  – State Diagrams
  – Activity Diagrams
  – Deployment Diagrams
• Some Conclusions

Package Diagrams

USC - Center for Software Engineering

- Helps in breaking down large software system into smaller pieces.
- Before OO, functional decomposition used to separate behavioral decomposition from data decomposition. This lead to a system decomposition which is incompatible with OO since here behavior and data cannot be separated.
- With OO, both forms of decompositions were merged and, thus, packages are really just a grouping mechanism for classes.
- Relationships between packages are similar to those of classes, however, package interrelationships should decrease coupling and increase cohesion.
Packages are self-contained entities like modules. Coupling should be minimized. This makes them particularly useful for testing and test scenarios (e.g., sequence diagrams).
State Diagrams

State Diagrams have been around for a long time, even before OO. They are very useful in presenting what states an object/class may go through in its lifetime.

Thus, in OO a state diagram represent the life of a single class or object. This is necessary because state diagrams are a non-OO design technique. Using state diagrams in OO on a system level may yield a functional system decomposition and not an OO one.

=> use state diagrams only with classes

=> If used on a higher level (e.g. to describe use cases) be careful when using its decomposition during design.
Another State Diagram

Another State Diagram for Order Class

How do we represent previous state diagram and current one together?

Authorized

[ payment OK ]

[ payment not OK ]

Rejected

Delivered

Concurrent State Diagram

Careful not to make a single class too complex
e.g. break up the class into Dispatching and Authorizing.
Activity Diagrams are the newest addition to UML and they are also unique in that they did not exist before. Activity Diagrams merge concepts from Event Diagrams, SDL state modeling, and Petri nets.

Activity Diagrams appear like enriched state diagrams, however, the meaning of components and connectors are not quite the same.

Activity diagrams cause a functional system breakdown and must be handled carefully. Nevertheless, through their emphasis on activities (like operators in classes), a OO interpretation of activity diagrams is much easier.
Outline

- UML Views
- Diagrams
  - Use Case Diagrams
  - Class Diagrams
  - Interaction Diagrams
  - Package Diagrams
  - State Diagrams
  - Activity Diagrams
  - Deployment Diagrams
- Some Conclusions

Deployment Diagrams

Deployment Diagrams show the physical dependencies between software and hardware components.
- Nodes represent computational units
- Components represent packages (subsystems) and
- Connectors represent communication paths.
Outline

- UML Views
- Diagrams
  - Use Case Diagrams
  - Class Diagrams
  - Interaction Diagrams
  - Package Diagrams
  - State Diagrams
  - Activity Diagrams
  - Deployment Diagrams
- Some Conclusions
Extending UML

Use Stereotypes to
- extend the notation of UML
- specialize the meaning of modeling elements

Use Object Constraint Language to
- extend the semantics of UML
- add additional constraints to modeling elements

=> extend to redefine
=> extend to specialize

The UML Meta-Model

Meta-Meta Model
Meta Model
Model
Objects
Sequence Calls

Meta-UML Structure
UML Structure
Static Structure
Dynamic Structure
Simplified UML Meta-Model

- UML notation is not very complex but there are many subtleties in interpreting things.
- UML is not well suited for precision (too many ambiguities).
- Nevertheless, UML is sufficient in modeling many (most) things. Often there is no need for strong formalism.

=> use UML during general development process
=> use something else (e.g. ADLs) when it is less suited